# A Debiasing Autoencoder for Recommender System

Teng Huang, Cheng Liang *Student Member, IEEE*, Di Wu, *Member, IEEE*, and Yi He, *Member, IEEE*

***Abstract*—The deep neural network (DNN)-based recommender system (RS) has drawn much attention recently and provided state-of-the-art results. Although many DNN-based RSs have been achieved, most focus on inventing a sophisticated DNN with a single-metric loss function to fit user behavior data. However, user behavior data are commonly collected from numerous users in complex scenarios, making various biases and outliers (outliers can be seen as the special bias) widely exist in the data. Unfortunately, prior DNN-based RSs only considered rather fragmented biases and lacked a comprehensive solution. To fill this gap, this paper proposes an AutoRec++ model to comprehensively address the various biases existed in user behavior data. Its main idea is to employ different combinations of preprocessing bias (PB) and training bias (TB) as well as $L_1$-norm and $L_2$-norm to form a multi-metric loss function-oriented Autoencoder. As such, AutoRec++ possesses the multi-merits of the PB's and TB's debiasing ability, the $L_1$-norm's robustness, and the $L_2$-norm's stability. By conducting extensive experiments on five benchmark datasets, we demonstrate that: 1) the incorporation of PB and TB can significantly boost Autoencoder's prediction accuracy and computational efficiency without structural change, and 2) our AutoRec++ achieves better prediction accuracy and robustness than both DNN-based and non-DNN-based state-of-the-art models. Besides, our AutoRec++ is more effective in processing sparser user behavior data. Our code is available at the link: *https://github.com/wudi1989/AutoRec_Plus*.**

***Index Terms*—Recommender System, Debias Methods, Representation Learning, Deep Neural Network, Biased Data Representation, Autoencoder.**

## I. Introduction

In this era of Big Data, the recommender system (RS) has been recognized as one of the most efficient and effective ways to address information explosion [1]–[5]. RS can not only facilitate users to filter desired information out of big data but also help service providers cater to users' tastes to make more profits [1], [6]. Therefore, multiple RS techniques have sprung up in the latest decade and have been widely applied to countless applications [2], [7], such as social networks (e.g., Facebook), E-commerce platforms (e.g., Amazon), and video-sharing platforms (e.g., Douyin and TikTok).

To implement an RS, the crux lies in exploring users' potential preferences on different items (e.g., news, short videos, music, movies, and commodities) from the observed user behavior data to make recommendations [8]–[13]. Yet user behavior data are commonly collected from numerous users in various scenarios, resulting in multiple biases and outliers widely existing in the collected data, which challenges the feature extraction of the methods[14]–[17].

For example, users' selection bias denotes that users may only rate items they are interested in, and those not interested in may not generate rating records. In addition, different rating preferences also exist among users, e.g., some users consider four out of five as an ordinary score, while others may consider that as the highest score. These kinds of biases from the user side are regarded as User Bias in our study. On the other hand, Item Bias from the item side also widely exists in the collected user behavior data. For instance, popular items tend to be more exposed to users than unpopular ones, resulting in imbalanced distributions of observed user behaviors regarding items. Furthermore, strict or malicious users may assign an extremely low or high rating to different items, causing outliers (outlier can be seen as the special bias). Since the user/item biases and outliers are the inherent characteristics of user behavior data, ignoring them will cause severe deterioration of recommendation performance [18]–[24].

Recently, with the rapid development of deep learning, deep neural network (DNN) has been widely applied to implementing RSs. So far, various DNN-based RSs have been proposed to achieve the state-of-the-art recommendation performance [10], [11], [25]–[29], where most of them pay much attention to inventing a sophisticated model to fit user behavior data better. As examples mentioned above, various user/item biases (including outliers) widely exist in the collected user behavior data. However, existing DNN-based RSs only consider rather fragmented biases and lack a comprehensive solution [18]–[24], [30]. Besides, they share a common essence that their loss function is $L_2$-norm-oriented, resulting in limited robustness as $L_2$-norm is sensitive to outliers [31], [32].

To fill this gap, this paper aims to comprehensively address the various biases (including outliers) in user behavior data for DNN-based RSs. To this end, we propose an AutoRec++ model by employing different combinations of preprocessing bias (PB) and training bias (TB) as well as $L_1$-norm and $L_2$-norm to form a multi-metric loss function-oriented Autoencoder. As such, AutoRec++ possesses the multi-merits of the PB's and TB's debiasing ability, the $L_1$-norm's robustness, and the $L_2$-norm's stability.

This paper has the following main contributions:

- It is the first study to take in-depth analyses regarding the effects of different combinations of PB and TB as well as $L_1$-norm and $L_2$-norm on the DNN-based RS.
- It proposes an AutoRec++ recommendation model that can comprehensively address the various biases (including outliers) existed in user behavior data.
- It conducts extensive empirical studies to evaluate the proposed AutoRec++, including characteristics analyses

This work is supported by the National Natural Science Foundation of China under grants 62176070 and 62002074. Teng Huang and Cheng Liang are *co-first authors* of this paper. Di Wu is the *corresponding author*.

Teng Huang and Cheng Liang are with the Institute of Artificial Intelligence and Blockchain, Guangzhou University, Guangzhou 510002, China (e-mail: huangteng1220@buaa.edu.cn, c_liang@e.gzhu.edu.cn).

Di Wu is with the College of Computer and Information Science, Southwest University, Chongqing 400715, China (e-mail: wudi.cigit@gmail.com).

He Yi is with the Department of Computer Science, Old Dominion University, Norfolk, Virginia 23529, USA (e-mail: yihe@cs.odu.edu).

and comparisons with state-of-the-art models.

Experimental results on five benchmark datasets demonstrate that: 1) Autoencoder's prediction accuracy and computational efficiency can be significantly boosted by incorporating the optimal PB and TB combination without structural change, 2) the self-adaptively weighted $L_1$-$L_2$-norm-oriented loss function can improve Autoencoder's robustness to outliers, and 3) our proposed AutoRec++ significantly outperforms both DNN-based and non-DNN-based state-of-the-art models in terms of prediction accuracy and robustness.

## II. Related Work

### A. DNN-based RS

Collaborative Filtering (CF) is one of the most popular and effective ways to implement an RS [2]. Its principle is to employ the historical user behavior data to exploit the similarity among users/items to predict users' potential preferences on items. Matrix factorization (MF) [8], [9], [33]–[35] is a representative approach of CF. It usually maps the user-item interaction matrix into two latent matrices to discover the similarity among users/items. However, such a shallow model is difficult to model the deep nonlinear relationships between users and items hidden in the complex user behavior data.

In comparison, due to DNN's powerful nonlinear learning ability originating from its deep learning structure, it has been widely applied to developing CF-based RSs [11], [25], [26], [28], [29]. For example, Autoencoder-based CF models [28], [29] are widely used in RSs tasks. They reduce the user-item interaction matrix to a low-dimension space to learn the similarity among users/items and make predictions. Besides, some models are similar to Autoencoder in structure but with different mechanisms, such as SparseFC [25], which parametrizes the weight matrices into low-dimension vectors to capture the features of user-item interaction matrix to make predictions. In addition, some hybrid DNN-based CF models have been proposed. For instance, NRR [11] maps user and item features into latent factors and then feeds them into a gated recurrent unit for tips generation. Glocal-K [26] combines Autoencoder with Radial Basis Function. After the pretraining and fine-tuning process, Glocal-K can predict the missing ratings by utilizing the global kernel. More recently, graph neural network (GNN)-based CF models [36], [37] have been constructed. They exploit users' preferences on items from a constructed user-item graph. For instance, IGMC [36] creates sub-graphs for each user-item pair by the two components in GNN to make predictions. Notably, despite these models' success in implementing an RS, they never comprehensively consider biases issues.

### B. Debiasing method

To address the different kinds of biases existing in user behavior data, various methods have been proposed [18]–[24]. Since missing ratings are commonly not random, data imputation with an asymmetric tri-training framework [24] is proposed to address the selection bias by imputing the missing ratings. By calculating the propensity rating of the interaction between users and items [21], the selection bias and rating bias can be reduced. Regarding the exposure bias, the exposure probability of unpopular items can be increased by weighting each observation with the inverse of its popularity [18]. Besides, oversampling the unpopular items [20] or introducing the exposure variable into the model [22] can also alleviate the exposure bias. The above biases are treated as linear biases and have been addressed in some shallow models [19]. Further, these linear biases are divided into User Bias and Item Bias, which can be addressed in the processing and training process [23], respectively. So far, existing DNN-based RSs only utilize partial debiasing methods and lack a comprehensive solution.

### C. Denoising method

Although previous studies spare their effort to address different biases through different methods, they all adopt the $L_2$-norm to form their loss function, which is highly sensitive to outliers [38]. To alleviate this issue, [32] proposes an $L_1$-norm-oriented loss function to improve the robustness of an RS. Yet, an $L_1$-norm-oriented model has muti-solutions due to the instability of $L_1$-norm. To aggregate both merits of stability and robustness, [31] and [38] propose to jointly utilize $L_1$-norm and $L_2$-norm to form the loss function in a single optimization framework. However, such $L_1$-and-$L_2$-norm-oriented loss function is built on the shallow MF framework, and its effectiveness on the DNN-based models is still unclear.

### D. Differences of our work

Although a lot of related studies have been done to address biases, they still have the following limitations:

- Most debiasing methods are designed for shallow MF-based CF models, which still cannot model complex nonlinear relationships between users and items.
- Most prior DNN-based RSs only consider partial biases and lack a comprehensive solution.
- Most prior DNN-based RSs are trained under a single-metric-oriented $L_2$-norm loss function, resulting in insufficient robustness when facing outliers.

In comparison, our proposed AutoRec++ is significantly different from the prior studies as follows:

- It can comprehensively address the various biases by incorporating the different combinations of PB and TB into a deep Autoencoder.
- It is robust to outliers by forming an $L_1$-and-$L_2$-norm-oriented loss function.

## III. Preliminaries

### A. Symbol Appointment

All the adopted symbols and the corresponding descriptions are listed in TABLE S.I in the Supplementary File.

### B. Definition of Recommendation Task

**Definition 1(User-item matrix):** Given a user set $M$ and an item set $N$, the matrix $Y \in \mathbb{R}$ with $|M|$ rows and $|N|$ column records the relationship between $M$ and $N$, i.e., each entry $y_{mn} \in Y(m \in M, n \in N)$ denotes the behavior data of user $m$ on item $n$, we set those unobserved entries as zero and observed ones as the observed value. Each user m can be

represented as the vector $\mathbf{y}^{(m)} = \{y_{m1}, \cdots, y_{m|N|}\} \in \mathbb{R}^{|N|}$, while each item $n$ can be represented as vector $\mathbf{y}^{(n)} = \{y_{1n}, \cdots, y_{|M|n}\} \in \mathbb{R}^{|M|}$. Binary matrix $D^{|M|\times|N|} \in \mathbb{R}$ distinguishes the observed and unobserved interaction in $Y$:

$$d_{mn} = \begin{cases} 1 \text{ if } y_{mn} \text{ is observed} \\ 0 \text{ otherwise} \end{cases}, \tag{1}$$

where $d_{mn}$ is a specific entry of matrix $D$.

**Definition 2(Problem):** There are two main tasks in RSs, i.e., rating and ranking prediction. This paper focuses on rating prediction because user/item biases are more common for this task. The problem of rating prediction is to learn a parametric model $f(\cdot)$ from the observed ratings of Y to make predictions for its unobserved ratings as follows:

$$f(M, N; \theta) \to Y, \tag{2}$$

where $\theta$ denotes the parameters of $f(\cdot)$. The objective function of $f(\cdot)$ is to minimize the following *Empirical Risk*

$$L(f) = \sum_{m\in M, n\in M} \epsilon\left(f(m, n; \theta), y_{mn}\right), \tag{3}$$

where $\epsilon(\cdot)$ denotes the error function measuring the distance between the prediction output$\hat{y}_{mn}$ from $f(m, n; \theta)$ and the ground true rating $y_{mn}$.

### C. AutoRec

AutoRec[28] is a representative DNN-based CF model. The user-based AutoRec (U-AutoRec) and the item-based AutoRec (I-AutoRec) are two variants of AutoRec. Formally, given a user-item matrix $Y$, AutoRec solves the same problem defined in the Eq. (3). It minimizes the following objective function:

$$\begin{aligned} L(f) = \sum_{y^{(q)}\in Y} \left\| \left(\mathbf{y}^{(q)} - f\left(\mathbf{y}^{(q)}; \theta\right)\right) \odot \mathbf{d}^{(q)} \right\|_{L_2}^2 \\ + \frac{\lambda}{2} \cdot \left( \|w_1\|_{L_2}^2 + \cdots + \|w_K\|_{L_2}^2 \right), \end{aligned} \tag{4}$$

where $\lambda > 0$ denotes the regularization factor to prevent AutoRec from overfitting, $\theta = \{w_1, \cdots, w_K, b_1, \cdots, b_K\}$ denotes the weighted term $w_k$ and intercept term $b_k$ of hidden layers, $k \in \{1, 2, \cdots, K\}$, $\|\cdot\|_{L_2}^2$ is the square of $L_2$-norm, $\mathbf{d}^{(q)}$ represents the $n_{th}$ column ($m_{th}$ row) in I-AutoRec (U-AutoRec) of index matrix $D$ respectively, $\odot$ is the Hadamard product, and $\mathbf{y}^{(q)}$ equals to item vector $\mathbf{y}^{(n)} = \{y_{1n}, \cdots, y_{|M|n}\}$ in I-AutoRec and user vector $\mathbf{y}^{(m)} = \{Y_{m1}, \cdots, Y_{m|N|}\}$ in U-AutoRec.

## IV. The Proposed AutoRec++

This section introduces the proposed AutoRec++. Fig 1 shows the architecture of AutoRec++, where the symbols are the same as those in Eq. (8) and the parameters within the grey wireframe are trainable. We adopt the blue cubes to represent PB and the grey cubes to represent TB. The training process is as follows: 1) subtract PB combinations $\psi$ from input data $\mathbf{y}^{(q)}$ to obtain the sub-input $\mathbf{y}^{(q)'}$, 2) process the sub-input $\mathbf{y}^{(q)'}$ with the model and add the output of last hidden layer $l_K$ with TB combinations $\phi$ to obtain the pre-output $\hat{\mathbf{y}}^{(q)'}$, and 3) add $\hat{\mathbf{y}}^{(q)'}$ with $\psi$ to obtain the final output of AutoRec++. Next,

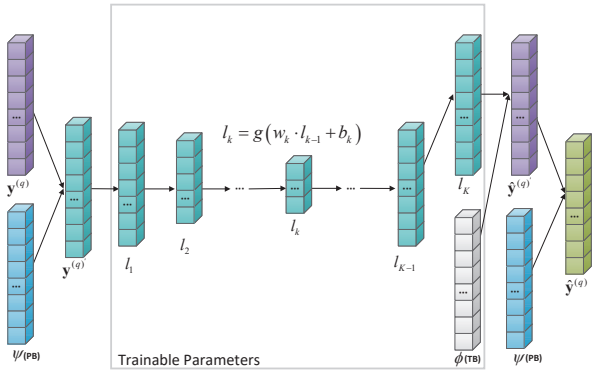



Fig. 1. The architecture of AutoRec++

we detail AutoRec++ which consists of three parts: calculation of PB and TB, incorporation of PB and TB, and denoised loss function.

### A. Calculation of PB and TB

**Preprocessing Bias.** The Preprocessing Bias (PB) combinations are obtained by preprocessing on the observed user behavior data before training. The PB contains three parts: $Pa$, $Pi^{(n)}$ and $Pu^{(m)}$, which represent the global average rating, item bias, and user bias, respectively. To form different combinations of PB, we use $z_1, z_2, z_3$ to control the value of $Pa, Pi^{(n)}, Pu^{(m)}$ separately. The global average rating $Pa \in \mathbb{N}^+$ denotes the statistical characteristics of observed ratings in $Y$. It can be obtained as follow:

$$Pa = z_1 \cdot \frac{\sum_{mn\in\Gamma} y_{mn}}{\sum_{mn\in\Gamma} d_{mn}}, \tag{5}$$

where $\Gamma \in \mathbb{R}^{|M|\times|N|}$ denotes the training set, i.e., a subset of the matrix $Y$ The preference of all users on an item $n$ can be estimated by $Pi^{(n)}$ with the following Equation:

$$Pi^{(n)} = z_2 \cdot \frac{\sum_{m\in\Gamma(n)} d_{mn} \cdot (y_{mn} - Pa)}{\omega_1 + \sum_{m\in\Gamma(n)} d_{mn}}, \tag{6}$$

where $\Gamma(n)$ represents the $n_{th}$ column of $\Gamma$, and $\omega_1$ represents the threshold constant related to the average rating of item $n$. The preference of a user on different items $Pu$ can be represented as:

$$Pu^{(m)} = z_3 \cdot \frac{\sum_{n\in\Gamma(m)} d_{mn} \cdot \left(y_{mn} - Pa - Pi^{(n)}\right)}{\omega_2 + \sum_{n\in\Gamma(m)} d_{mn}} \tag{7}$$

All the possible combinations of PB are summarized in TABLE I. Furthermore, we let $Pi$ and $Pu$ represent the PB vector of all items and all users, respectively.

**Training Bias.** Unlike PB, the Training Bias (TB) processes are synchronized with the training process. The TB comprises two parts: $Ti^{(n)}$ and $Tu^{(m)}$, which can excavate the preference of user $m$ and item $n$, respectively, during the training process. Similarly, we utilize $z_4, z_5$ to control the value of $Ti^{(n)}, Tu^{(m)}$ to form different TB combinations and $Ti, Tu$ to represent the TB vector of all items and all users. All possible combinations of TB are summarized in TABLE II.

### B. Incorporation of PB and TB

By incorporating various combinations of PB and TB into the representative AutoRec, we have AutoRec++. However, in

TABLE I. ALL POSSIBLE COMBINATIONS OF PB.

| | PB.1 | PB.2 | PB.3 | PB.4 | PB.5 | PB.6 | PB.7 | PB.8 |
|---|---|---|---|---|---|---|---|---|
| value of weights | $z_1=0$<br>$z_2=0$<br>$z_3=0$ | $z_1=1$<br>$z_2=1$<br>$z_3=1$ | $z_1=0$<br>$z_2=1$<br>$z_3=1$ | $z_1=1$<br>$z_2=0$<br>$z_3=1$ | $z_1=1$<br>$z_2=1$<br>$z_3=0$ | $z_1=0$<br>$z_2=0$<br>$z_3=1$ | $z_1=0$<br>$z_2=1$<br>$z_3=0$ | $z_1=1$<br>$z_2=0$<br>$z_3=0$ |

TABLE II. ALL POSSIBLE COMBINATIONS OF TB.

| | TB.1 | TB.2 | TB.3 | TB.4 |
|---|---|---|---|---|
| value of weights | $z_4=0$<br>$z_5=0$ | $z_4=1$<br>$z_5=1$ | $z_4=1$<br>$z_5=0$ | $z_4=0$<br>$z_5=1$ |

the practical situation, the user can only explore some of the possible bias combinations. Thus, we first train AutoRec++ with the stable $L_2$-norm loss function for a fair competition to obtain the overall optimal bias combination. Formally, we use $l_k$ to represent the $k_{th}, k \in \{1, 2, ..., K\}$, hidden layer of AutoRec++, and $w_k$ and $b_k$ denote the weighted term and intercept term of $l_k$ respectively. Thus, the prediction function of AutoRec++ can be written as follows:

$$
\begin{aligned}
&\mathbf{y}^{(q)'} = \mathbf{y}^{(q)} - \psi \\
&l_1 = g\left(w_1 \cdot \mathbf{y}^{(q)'} + b_1\right) \\
&l_k = g\left(w_k \cdot l_{k-1} + b_k\right), k = 2, \ldots, K-1 \\
&\hat{\mathbf{y}}^{(q)'} = w_K \cdot l_{K-1} + b_K + \phi \\
&\hat{\mathbf{y}}^{(q)} = f\left(\mathbf{y}^{(q)}; \theta, \phi, \psi\right) = h\left(l_K + \psi\right) \\
&\begin{cases}
\text{I-AutoRec ++:} & \begin{cases} \mathbf{y}^{(q)} = \mathbf{y}^{(n)} = \left\{y_{1n}, \ldots, y_{|M|n}\right\} \\ \psi = Pa + Pi^{(n)} + Pu \\ \phi = Ti^{(n)} + Tu \end{cases} \\
\text{U-AutoRec ++} & \begin{cases} \mathbf{y}^{(q)} = \mathbf{y}^{(m)} = \left\{y_{m1}, \ldots, y_{m|N|}\right\} \\ \psi = Pa + Pi + Pu^{(m)} \\ \phi = Ti + Tu^{(m)} \end{cases}
\end{cases},
\end{aligned}
\tag{8}
$$

where $\mathbf{y}^{(q)'}$ represents the input after subtracting PB, $\hat{\mathbf{y}}^{(q)'}$ is the sub-output, $\hat{\mathbf{y}}^{(q)}$ is the final output of Autorec++, $\psi$ and $\phi$ are different combinations of PB and TB in TABLE I and TABLE II, $\Theta = \{w_1, \cdots, w_K, b_1, \cdots, b_K\}$ are the learnable parameters, $h(\cdot)$ and $g(\cdot)$ represent the mapping function and activation function respectively. The objective function for AutoRec++ is to minimize the following Equation:

$$
\begin{cases}
\text{I-AutoRec++:} & \begin{cases} L(f) = \sum\limits_{\substack{\mathbf{y}^{(n)} \in Y \\ \mathbf{d}^{(n)} = D}} \left(\left(\mathbf{y}^{(n)} - \hat{\mathbf{y}}^{(n)}\right) \odot \mathbf{d}^{(n)}\right)^2 \\ + \frac{\lambda_1}{2} \cdot \left(\sum_{k=1}^{K} \left(w_k\right)^2\right) + \frac{\lambda_2}{2} \cdot \left(Ti^{(n)} + Tu\right)^2 \end{cases} \\
\text{U-AutoRec++:} & \begin{cases} L(f) = \sum\limits_{\substack{\mathbf{y}^{(m)} \in Y \\ \mathbf{d}^{(m)} \in D}} \left(\left(\mathbf{y}^{(m)} - \hat{\mathbf{y}}^{(m)}\right) \odot \mathbf{d}^{(m)}\right)^2 \\ + \frac{\lambda_1}{2} \cdot \left(\sum_{k=1}^{K} \left(w_k\right)^2\right) + \frac{\lambda_2}{2} \cdot \left(Ti + Tu^{(m)}\right)^2 \end{cases}
\end{cases},
\tag{9}
$$

where $\lambda_1, \lambda_2$ are the regularization rate hyperparameters. To minimize the function $L(f)$ and train the learnable parameters $\Theta = \{w_1, b_1, \cdots, w_K, d_k, Tu, Ti\}$ above, we adopted Adam [39] to adapt the learning rate automatically to train $\Theta$.

### C. Denoised loss function

As the previous description, the $L_2$-norm loss function is sensitive to the outliers in the user behavior data (e.g., a user always assigns an extraordinarily high or low rating to different items), and the $L_1$-norm loss function is less sensitive to outliers but lacks stability during training. To aggregate both merits of robustness and stability, we jointly utilize them to form the self-adaptively weighted $L_1$-and-$L_2$-norm-oriented loss function. Thus, the objective function with $L_1$-and-$L_2$-norm-oriented loss function of AutoRec++ becomes:

$$
\begin{cases}
\text{I-AutoRec++:} & \begin{cases} L(f) = \sum\limits_{\substack{\mathbf{y}^{(n)} \in Y \\ \mathbf{d}^{(n)} = D}} \gamma_1 \cdot \left|\left(\mathbf{y}^{(n)} - \hat{\mathbf{y}}^{(n)}\right) \odot \mathbf{d}^{(n)}\right| \\ + \gamma_2 \cdot \left(\left(\mathbf{y}^{(n)} - \hat{\mathbf{y}}^{(n)}\right) \odot \mathbf{d}^{(n)}\right)^2 \\ + \frac{\lambda_1}{2} \cdot \left(\sum_{k=1}^{K} \left(w_k\right)^2\right) + \frac{\lambda_2}{2} \cdot \left(Ti^{(n)} + Tu\right)^2 \end{cases} \\
\text{U-AutoRec++:} & \begin{cases} L(f) = \sum\limits_{\substack{\mathbf{y}^{(m)} \in Y \\ \mathbf{d}^{(m)} \in D}} \gamma_1 \cdot \left|\left(\mathbf{y}^{(m)} - \hat{\mathbf{y}}^{(m)}\right) \odot \mathbf{d}^{(m)}\right| \\ + \gamma_2 \cdot \left(\left(\mathbf{y}^{(m)} - \hat{\mathbf{y}}^{(m)}\right) \odot \mathbf{d}^{(m)}\right)^2 \\ + \frac{\lambda_1}{2} \cdot \left(\sum_{k=1}^{K} \left(w_k\right)^2\right) + \frac{\lambda_2}{2} \cdot \left(Ti + Tu^{(m)}\right)^2 \end{cases}
\end{cases},
\tag{10}
$$

where $\gamma_1$ and $\gamma_2$ are the aggregation weights self-adaptively control the effects of $L_1$-norm and $L_2$-norm, respectively. Notably, we set $\gamma_1 + \gamma_2 = 1$, and $\gamma_1, \gamma_2 \geq 0$ to maintain the magnitude of loss. The main idea of the aggregation strategy is to increase $\gamma_1$ if the partial loss of $L_1$-norm is lower than $L_2$-norm, otherwise increasing $\gamma_2$. For a better presentation, we let $\delta_1^t$ and $\delta_2^t$ denote the partial losses of $L_1$-norm and $L_2$-norm at the $t_{th}$ training iteration, and $\delta_{1,2}^t$ is the sum of $\delta_1^t$ and $\delta_2^t$, and $A_1^t$, $A_2^t$, $A_{1,2}^t$ is the accumulative loss of $\delta_1^t$, $\delta_2^t$, $\delta_{1,2}^t$ during $t$ training iteration respectively:

$$
\begin{aligned}
\delta_{1,2}^t &= \gamma_1^t \cdot \delta_1^t + \gamma_2^t \cdot \delta_2^t \\
A_1^t = \sum_{j=1}^{t} \delta_1^j, \quad A_2^t &= \sum_{j=1}^{t} \delta_2^j, \quad A_{1,2}^t = \sum_{j=1}^{t} \delta_{1,2}^j
\end{aligned}
\tag{11}
$$

Based on the definition above, we have *Theorem* 1. ***Theorem* 1**: Assuming that $\delta_1^t$ and $\delta_2^t$ land in the scale of $[0, 1]$, and if $\gamma_1^t$ and $\gamma_2^t$ of the $t_{th}$ iteration meet the following conditions:

$$
\gamma_1^t = \frac{e^{-\mu A_1^{t-1}}}{e^{-\mu A_1^{t-1}} + e^{-\mu A_2^{t-1}}}, \quad \gamma_2^n = \frac{e^{-\mu A_2^{t-1}}}{e^{-\mu A_1^{t-1}} + e^{-\mu A_2^{t-1}}},
\tag{12}
$$

where the hyperparameter $\mu$ controls their learning rates, and $T$ is the maximum training iteration. Then the subsequent inequality holds:

$$
A_{1,2}^T \leq \min\{A_1^T, A_2^T\} + \frac{\ln 2}{\mu} + \frac{\mu T}{8}
\tag{13}
$$

By setting $\mu$ as $(1/\ln T)^{1/2}$, the upper bound becomes: $\min\{A_1^T, A_2^T\} + \ln 2(\ln T)^{1/2} + T/((8 \ln T)^{1/2})$, which is linearly related to the maximum training iteration. Based on *Theorem* 1, we further propose *Proposition* 1.

***Proposition* 1**: Given $\mu = \sqrt{1/\ln T}$, and if $A_1^T > A_1^T$, the subsequent inequality holds:

$$
A_{1,2}^T \leq A_2^T + const < A_1^T + const,
\tag{14}
$$

otherwise:

$$
A_{1,2}^T \geq A_1^T + const < A_2^T + const,
\tag{15}
$$

where $\lim_{T \to \infty} const = 19.45$.

***Remark* 1**: Based on *Proposition* 1, by setting $\mu = \sqrt{1/\ln T}$, $A_{1,2}^T$ is bounded by $A_1^T + const$ and $A_2^T + const$. Thus, by training AutoRec++ under the self-adaptively aggregated $L_1$-and-$L_2$-norm loss function, its accumulative loss will always be lower than that training under the single metric-based $L_1$-norm or $L_2$-norm loss function. Hence, *Proposition* 1 shows that the self-adaptively $\gamma_1$ and $\gamma_2$ in Eq. (12) can

independently control the effect of robustness (relying on $L_1$-norm) and stability (relying on $L_2$-norm), making the targeted model's performance be better than that trained separately under these two loss function.

The proofs of *Theorem* 1 and *Proposition* 1 are detailed in Section III of the Supplementary File.

### D. Time and Space Complexity of AutoRec++

Specifically, feeding a hidden unit of I-AutoRec++ with $|M|$ inputs needs $|M|$ multiplications plus $|M|+1$ additions of the intercept term and activation function, which are proportional to $O(|M|)$. Supposed I-AutoRec++ has $x$ hidden units in the first layer, with $|M|$ inputs each, the big-o of the first layer is $x \times O(|M|)$. In this paper, we set the number of hidden layers $K$ as two in AutoRec++. So, the time complexity of I-AutoRec++ is represented as $x \times O(|M|) + |M| \times x \times O(|M|)$ equals to $O((|M|^2) \times x)$. If we take $x$ as a constant, the time complexity of I-AutoRec++ is $O(|M|^2)$. Similarly, the time complexity of U-AutoRec++ is deduced in the same way, which equals to $O((|N|^2) \times x)$ and $O(|N|^2)$ if $x$ is considered as a constant.

AutoRec++ only needs to reserve the input matrix and other variants with the same dimension that records additional information. And I-AutoRec needs $O(|M| \times K$ space to store its parameter. So, the space complexity of I-AutoRec is $O(|M| \times |N| + |M| \times K)$. However, $N$ is usually much greater than $K$, so the space complexity of I-AutoRec++ is $O(|M| \times |N|)$, and the same for U-AutoRec.

## V. Experiments

In the subsequent experiments, we aim to answer the following research questions (RQs):

- RQ.1. How do the preprocessing bias (PB) and training bias (TB) combinations influence the performance of AutoRec?
- RQ.2. Does the proposed AutoRec++ outperform state-of-the-art models in rating prediction and robustness?
- RQ.3. How does the number of hidden units influence the performance of AutoRec++?

### A. General Settings

**Datasets.** Five frequently used benchmark datasets are chosen to conduct the following experiments. They are real datasets from different fields, including e-commerce and movie review sites. TABLE S.II. in the Supplementary File summarizes their details. MovieLens_1M, MovieLens_100k and MovieLens_HetRec are collected from the movie recommendation website MovieLens[1]. Yahoo is collected from the Yahoo website[2]. Douban is collected from [40][3]. We adopt three kinds of train-test settings, i.e., 80%-20%, 50%-50%, and 20%-80%, to evaluate the different sparsity ratios of user behavior data.

**Evaluation Metrics.** Predicting the missing ratings of the user-item matrix is one of the two recommendation tasks.

[1]https://grouplens.org/datasets/movielens/

[2]https://webscope.sandbox.yahoo.com/catalog.php?datatype=r

[3]https://github.com/fmonti/mgcnn

TABLE III. The winning cases of different PB and TB combinations on 5 datasets with 3 different split ratios

| | PB.1 | PB.2 | PB.3 | PB.4 | PB.5 | PB.6 | PB.7 | PB.8 |
|---|---|---|---|---|---|---|---|---|
| TB.1 | 0 | 6 | 0 | 0 | 0 | 0 | 0 | 0 |
| TB.2 | 2 | 28 | 4 | 2 | 2 | 0 | 2 | 0 |
| TB.3 | 0 | 3 | 0 | 7 | 1 | 0 | 0 | 0 |
| TB.4 | 0 | 1 | 0 | 2 | 0 | 0 | 0 | 0 |

The cases in each bias combination that achieved the best accuracy in TABLEs. S.IV-S.VI in Section V of the Supplementary File.

To evaluate the accuracy of missing rating prediction, root mean square error (RMSE) and mean absolute error (MAE) are adopted as the evaluation metrics:

$$\text{RMSE} = \sqrt{\frac{\sum_{m,n\in\Gamma'} \left((y_{mn} - \hat{y}_{mn}) \cdot d_{mn}\right)^2}{\sum_{m,n\in\Gamma'} d_{mn}}} \tag{16}$$

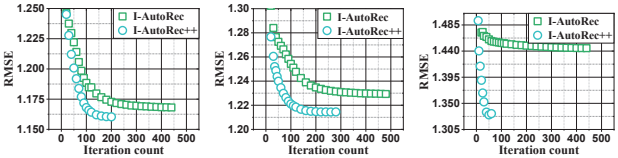
$$\text{MAE} = \frac{\sum_{m,n\in\Gamma'} |(y_{mn} - \hat{y}_{mn}) \cdot d_{mn}|_{abs}}{\sum_{m,n\in\Gamma'} d_{mn}} \tag{17}$$

where $\Gamma'$ denotes the testing set, and $|\cdot|_{abs}$ denotes the absolute value of a given number. **Baselines.** The proposed AutoRec++ is compared with seven state-of-the-art models, including one original model (AutoRec [28]), four deep-learning models (SparseFC [25], GLocal-K [26], IGMC [36], and NRR [11]), and two latent factor analysis-based (LFA-based) models (FML [8] and MF [9]). Table S.III in Section V of the Supplementary File gives a brief description of these competitors.

**Implementations Details.** Learning rate, batch size, and regularization rate are fine-tuned for all the models to achieve their own optimal prediction accuracy. The training procedure of a model will terminate if its training iterations reach the preset threshold (i.e., 500), and the best result will be selected. All the experiments are run on a GPU server that has 2 2.4GHz Xeon Gold 6240R with 24 cores, 376.40GB RAM, and 4 Tesla V100 GPUs.

### B. Influence of PB and TB (RQ.1)

*1) Influence of different PB and TB combinations:* The complete experimental result of AutoRec at different PB and TB combinations on five datasets with three different train-test ratios are recorded in TABLEs. S.IV-S.VI in Section V the Supplementary File. To better analyze these results, we make a statistical analysis of the winning cases in TABLE III, which counts how many cases the lowest RMSE/MAE a combination reaches. We use fixed hyperparameters fine-tuned for AutoRec in each dataset and the $L_2$-norm loss function to train the model for stable results. From TABLEs. S.IV-S.VI TABLE III,




(a) D4 with 80%-20% train-test ratio (b) D4 with 50%-50% train-test ratio (c) D4 with 20%-80% train-test ratio

Fig. 2. The Scatter plots of RMSE of AutoRec and AutoRec++ on D4 with 3 different train-test ratios during the training process.

we have the following important observations:

- The separate utilization of PB and TB combinations cannot offer sufficient accuracy improvement. From column PB.1 and row TB.1, we find that when PB and TB are used separately, they can only reach two and six best accuracy cases in AutoRec, respectively. But as PB and TB start to combine, e.g., PB.2-TB.2 PB.2-TB.3, it achieves more optimal accuracy cases.
- The combination of PB.2 and TB.2 reaches the twenty-eight lowest RMSE/MAE cases in AutoRec, better than other combinations.

We also integrated different bias combinations into the commonly used matrix factorization (MF)-based model to obtain optimal results and compare the accuracy of debiased MF-based RS and DNN-based RS. TABLE IV records their result on D1 to D5 under the 80%-20% train-test ratio, and the complete results are recorded on TABLE. S.VII in Section V of the supplementary file. However, there is no specific bias combination that shows an outstanding performance in MF, so we choose their optimal result as the final result, in which we have the following observations:

- The results of the debiased MF-based model are inferior to AutoRec++, and it even fails to offer higher accuracy in some cases. For example, in D2 and D3, the accuracy of the debiased MF is slightly lower than the original MF.
- The performance improvement in other datasets is not as pronounced as the optimal bias combination has on the DNN-based AutoRec.

Thus, we believe that the MF-based model is too shallow that offers a narrow retrofit performance margin. Even after incorporating different combinations, MF's nonlinear representation ability for the complex relationship between items and users is still inferior to DNN-based models.

**Summary.** These observations demonstrate that PB.2 and TB.2 reach the highest accuracy in AutoRec, and users can only explore some possible combinations in the practical application. Thus, we regard the PB.2 and TB.2 combination as the optimal bias combination. Since I-AutoRec can achieve a better performance than U-AutoRec, we integrate it with the optimal PB and TB combination to form AutoRec++. Besides, we show that the MF-based model is insufficient to represent the complex user-item relationship compared to the DNN-based model.

TABLE IV. THE PERFORMANCE COMPARISON BETWEEN MF-BASED AND DNN-BASED RS AND THEIR DEBIASED VERSION.

| Dataset | Metric | MF | debiased-MF | AutoRec | AutoRec++ |
|---|---|---|---|---|---|
| D1 | RMSE | 0.851 | 0.845 | 0.843 | 0.838 |
| | MAE | 0.669 | 0.664 | 0.664 | 0.655 |
| D2 | RMSE | 0.903 | 0.907 | 0.893 | 0.882 |
| | MAE | 0.711 | 0.709 | 0.703 | 0.691 |
| D3 | RMSE | 0.751 | 0.754 | 0.749 | 0.733 |
| | MAE | 0.567 | 0.569 | 0.569 | 0.556 |
| D4 | RMSE | 1.200 | 1.185 | 1.168 | 1.160 |
| | MAE | 0.934 | 0.905 | 0.903 | 0.893 |
| D5 | RMSE | 0.737 | 0.727 | 0.740 | 0.729 |
| | MAE | 0.586 | 0.571 | 0.584 | 0.574 |

*2) Influence with the optimal PB and TB combination:* To evaluate the influence of optimal PB and TB combination on DNN-based RS, we compare the training processes of AutoRec and AutoRec++. Part of the experimental results are shown in Fig. 2, and the rest are shown in Figs. S1,S2 in Section V of the Supplementary File, where we have the following observations:

- The optimal PB and TB combination can accelerate the training process of AutoRec. The optimal PB and TB combination can steeply reduce the training time of AutoRec++ to 20% to 50% of the original model.
- As the proportion of training data reduces, the reduction of training time and improvement in prediction accuracy becomes more pronounced.

We further discuss the accuracy improvement, TABLE V presents the percentages of RMSE/MAE drop (i.e., the accuracy improvement rate) after incorporating the optimal PB and TB combination into AutoRec, where we have the following observations:

- As the training data proportion decreases, the accuracy improvement percentage generally increases. But, in some cases, for example, on D2, the accuracy improvement rate of 50%-50% train-test ratio is lower than that of 80%-20%.
- When the division ratio of the training set is reduced to 20%, the reduction percentage of RMSE/MAE is rapidly increased, even reaching 9.42% on D4.

**Summary.** This set of experiments verifies that by incorporating the optimal PB and TB into the original AutoRec, its computational efficiency and prediction accuracy have been greatly improved. Such improvements are more effective with sparser datasets.

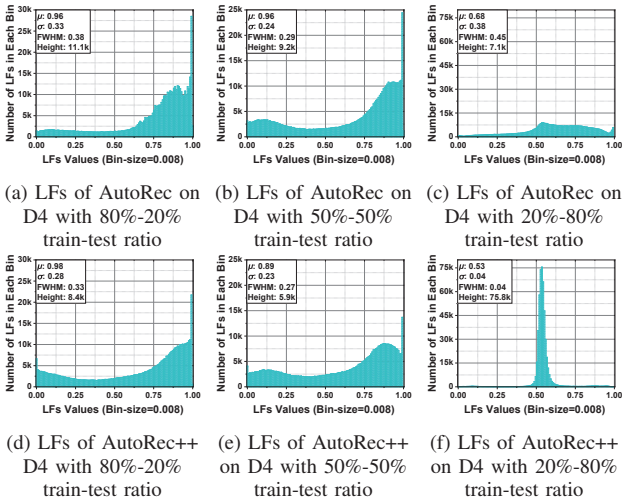
*3) Influence on the distribution of latent factors:* To better understand how the optimal PB and TB combinations influence AutoRec, we visualize the Encoder output (i.e., Latent Factors, LFs) distribution of AutoRec and AutoRec++. Fig. 3 shows part of the experimental results, and the remaining parts are shown in Figs. S3-S5 in Section V of the Supplementary File. To analyze these figures, we adopt a Gaussian function

$$h(e) = \xi_1 + \frac{\xi_2}{\sigma\sqrt{\pi/2}} e^{-2\frac{(e-\mu)^2}{\sigma^2}} \tag{18}$$

to fit them, where e is the value of LFs, h(e) is the number of LFs in each bin, $\xi_1$ and $\xi_2$ are the constant, $\mu$ is the expectation, and $\sigma$ is the standard deviation. The full width at half maximum (FWHM) and height of the Gaussian curve is measured. All the statistical parameters mentioned above

TABLE V. THE PERCENTAGE OF RMSE/MAE DROP OF AUTOREC AFTER INTEGRATION WITH OPTIMAL PB AND TB COMBINATION

| train-test | Metric | D1 | D2 | D3 | D4 | D5 |
|---|---|---|---|---|---|---|
| 80%-20% | RMSE | 0.59% | 1.23% | 2.14% | 0.68% | 1.49% |
| | MAE | 1.36% | 1.71% | 2.11% | 1.11% | 1.71% |
| 50%-50% | RMSE | 1.16% | 0.76% | 2.71% | 1.06% | 3.70% |
| | MAE | 1.90% | 1.24% | 3.38% | 1.36% | 4.19% |
| 20%-80% | RMSE | 1.54% | 2.81% | 2.82% | 9.42% | 4.25% |
| | MAE | 2.35% | 3.66% | 3.35% | 2.09% | 4.59% |




(a) LFs of AutoRec on D4 with 80%-20% train-test ratio

(b) LFs of AutoRec on D4 with 50%-50% train-test ratio

(c) LFs of AutoRec on D4 with 20%-80% train-test ratio

(d) LFs of AutoRec++ D4 with 80%-20% train-test ratio

(e) LFs of AutoRec++ on D4 with 50%-50% train-test ratio

(f) LFs of AutoRec++ on D4 with 20%-80% train-test ratio

Fig. 3. The distribution histogram of LFs of AutoRec and AutoRec++ on D4 with 3 different train-test ratios.

are inserted into the subsequent figures. Additionally, since the sigmoid function activates the output of the Encoder, all output values are landing between 0 and 1. From Figs. 3, S3-S5, we have the following observations:

- Compared to the original model, the LFs distribution of AutoRec++ is more concentrated in the middle after combining with the optimal bias combination, which is closer to the original data distribution.
- The concentrated effect of the optimal bias combination on LFs becomes more pronounced as the proportion of training data drops. From Fig. 3(c) and Fig. 3(f), as the proportion of training data decreases to 20%-80%, the distribution of LFs of the intermediate suddenly and sharply rises. The same is true for other graphs.

**Summary.** These observations reveal that the optimal PB and TB combination can change the distribution of LFs of AutoRec. Such changes are more pronounced as the ratio of training data decreases. According to the principle of ensemble [41], such changes are beneficial for boosting a base model.

*4) Influence on debiasing ability:* We compare the debiasing ability between AutoRec and AutoRec++ by rounding their prediction ratings to test the errors. Fig. 4 records their number of prediction errors on D1 to D5, from which we have the following observation:

- AutoRec++ significantly outperforms AutoRec in predicting extreme values (1 and 5). For instance, in D4, the prediction error of AutoRec++ on the rating 1 or 5 is extremely lower than AutoRec.
- Since AutoRec does not consider addressing biases, its prediction is more inclined to the average, i.e., 2 or 3. Thus AutoRec is slightly better than AutoRec++ in predicting the median value.

**Summary.** With minor scarification of performance in predicting average ratings, AutoRec++ shows better prediction accuracy in extreme values, which demonstrates that AutoRec++ possesses a better debiasing ability to capture users' preferences on most/least favorite items.

### C. Performance Comparison (RQ.2)

*1) Comparison of prediction accuracy:* TABLE VI records the prediction accuracy of all competitors on D1 to D5 under three kinds of train-test settings. To better analyze these results, we make a statistical analysis of the loss/tie/win, the Wilcoxon signed-ranks test [42], and the Friedman test [42]. The loss/tie/win counts how many cases AutoRec++, trained under the $L_1$-and-$L_2$-norm loss function, has higher/same/lower RMSE/MAE than each of the compared models on all datasets, respectively. The Wilcoxon signed-ranks test is a nonparametric pairwise comparison method. It checks whether the prediction accuracy of AutoRec++ is significantly higher than each comparison model by the level of significance $p$-value. The Friedman test compares the performance of multiple models on multiple datasets simultaneously by the F-rank value. Lower F-rank values indicate a higher prediction accuracy. The statistical results of the loss/tie/win, the Wilcoxon signed-ranks test, and the Friedman test are presented in the third-to-last, second-to-last, and last row of TABLE VI. From TABLE VI, we have the subsequent observations:

- AutoRec++ reaches the lowest RMSE/MAE in most cases. It has only lost 23 times and tied 2 times in the comparison. Specifically, the value of loss/tie/win is 23/2/185.
- All the $p$-values are lower than the significance level of 0.05, and it achieves the lowest F-rank among all the models, indicating that AutoRec++ has a better prediction accuracy than other competitors.

**Summary.** These observations demonstrate that AutoRec++ can outperform the comparison models in terms of missing rating prediction of the user behavior data.

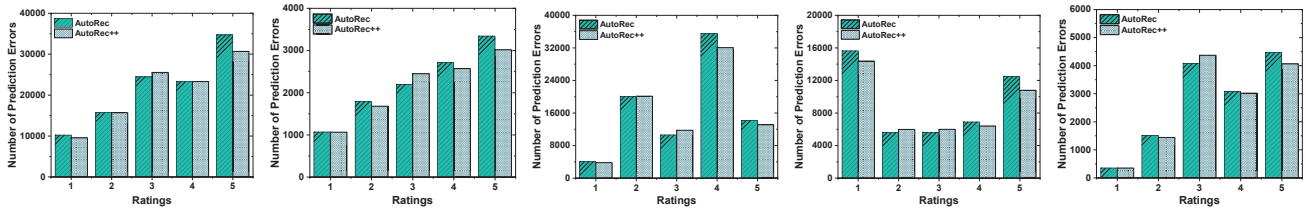
*2) Comparison of computational efficiency:* Fig. 5 and Fig. S6 in Section V of the Supplementary File are the histogram graphs that record the total time cost to reach the best testing accuracy of models. To better demonstrate the computational efficiency of these competitors, we set the maximum value of the y-axis to 3000 and 2000 in these histogram graphs. The parts that are beyond the maximum value will not be displayed. From Figs. 5,S6 we have the following observations:

- The GNN-based model consumes much more computational resources and time than other models due to its complexity of structure. In Figs. 5,S6, we found that IGMC's time consumption is higher than 3000 sec.
- The matrix factorization models are faster than DNN-based models in most cases because they only train at the observed data. Significantly, MF outperforms all models in training time consumption.
- The computational time consumed by AutoRec++ is landing between matrix factorization models and GNN-based models. Furthermore, it achieves the lowest time consumption among all the DNN-based models.

**Summary.** These observations reveal that the computational efficiency of AutoRec++ is higher than other DNN-based and GNN-based models, but slightly lower than matrix factorization models.

This article has been accepted for publication in IEEE Transactions on Consumer Electronics. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TCE.2023.3281521






(a) D1 (b) D2 (c) D3 (d) D4 (e) D5

Fig. 4. The prediction errors of different ratings on D1 to D5 with 80%-20% train-test ratio.

TABLE VI. THE COMPARISON RESULTS OF THE PREDICTION ACCURACY OF AUTOREC++ AND ITS COMPETITORS, INCLUDING THE LOSS/TIE/WIN COUNTS, WILCOXON SIGNED-RANKS TEST, AND FRIEDMAN TEST

| Dataset | | Metric | MF | AutoRec | NRR | SparseFC | IGMC | FML | Glocal-k | AutoRec++ (ours) |
|---|---|---|---|---|---|---|---|---|---|---|
| 80%-20% train-test | D1 | RMSE | 0.851● | 0.843● | 0.879● | 0.833○ | 0.864● | 0.845● | 0.832○ | 0.834 |
| | | MAE | 0.669● | 0.664● | 0.688● | 0.650● | 0.677● | 0.663● | 0.649● | 0.648 |
| | D2 | RMSE | 0.903● | 0.893● | 0.919● | 0.895● | 0.908● | 0.897● | 0.884● | 0.881 |
| | | MAE | 0.711● | 0.703● | 0.719● | 0.703● | 0.714● | 0.709● | 0.691● | 0.688 |
| | D3 | RMSE | 0.751● | 0.749● | 0.776● | 0.744● | 0.768● | 0.749● | 0.742● | 0.731 |
| | | MAE | 0.567● | 0.569● | 0.585● | 0.563● | 0.578● | 0.569● | 0.561● | 0.550 |
| | D4 | RMSE | 1.200● | 1.168● | 1.222● | 1.177● | 1.131○ | 1.162● | 1.183● | 1.164 |
| | | MAE | 0.934● | 0.903● | 0.944● | 0.895● | 0.858○ | 0.923● | 0.886● | 0.877 |
| | D5 | RMSE | 0.737● | 0.740● | 0.724○ | 0.740● | 0.747● | 0.759● | 0.734● | 0.730 |
| | | MAE | 0.586● | 0.584● | 0.570○ | 0.584● | 0.586● | 0.594● | 0.577● | 0.572 |
| 50%-50% train-test | D1 | RMSE | 0.871● | 0.865● | 0.885● | 0.855● | 0.888● | 0.864● | 0.855● | 0.854 |
| | | MAE | 0.686● | 0.683● | 0.695● | 0.670● | 0.699● | 0.681● | 0.671● | 0.666 |
| | D2 | RMSE | 0.935● | 0.918● | 0.943● | 0.929● | 0.936● | 0.928● | 0.912 | 0.912 |
| | | MAE | 0.740● | 0.726● | 0.742● | 0.733● | 0.736● | 0.737● | 0.715● | 0.714 |
| | D3 | RMSE | 0.810● | 0.775● | 0.790● | 0.764● | 0.789● | 0.769● | 0.770● | 0.752 |
| | | MAE | 0.617● | 0.591● | 0.595● | 0.580● | 0.596● | 0.586● | 0.583● | 0.567 |
| | D4 | RMSE | 1.243● | 1.229● | 1.252● | 1.265● | 1.188○ | 1.210○ | 1.258● | 1.217 |
| | | MAE | 0.994● | 0.954● | 0.960● | 0.966● | 0.921○ | 0.967● | 0.948● | 0.930 |
| | D5 | RMSE | 0.755○ | 0.783● | 0.751○ | 0.799● | 0.775● | 0.822● | 0.760● | 0.756 |
| | | MAE | 0.602● | 0.621● | 0.593○ | 0.634● | 0.614● | 0.675● | 0.600● | 0.594 |
| 20%-80% train-test | D1 | RMSE | 0.918● | 0.912● | 0.915● | 0.915● | 0.926● | 0.909● | 0.935● | 0.898 |
| | | MAE | 0.727● | 0.722● | 0.721● | 0.724● | 0.733● | 0.722● | 0.739● | 0.701 |
| | D2 | RMSE | 1.006● | 0.996● | 0.974● | 0.996● | 0.967○ | 0.987● | 1.047● | 0.973 |
| | | MAE | 0.803● | 0.793● | 0.771● | 0.793● | 0.761○ | 0.793● | 0.837● | 0.763 |
| | D3 | RMSE | 0.815● | 0.817● | 0.812● | 0.813● | 0.823● | 0.811● | 0.813● | 0.793 |
| | | MAE | 0.621● | 0.626● | 0.612● | 0.619● | 0.623● | 0.623● | 0.620● | 0.599 |
| | D4 | RMSE | 1.404● | 1.444● | 1.342○ | 1.472● | 1.287○ | 1.312○ | 1.394● | 1.366 |
| | | MAE | 1.176● | 1.099● | 1.053 | 1.118● | 1.030○ | 1.064● | 1.041○ | 1.053 |
| | D5 | RMSE | 0.776○ | 0.824● | 0.781○ | 0.822● | 0.822● | 0.848● | 0.801● | 0.794 |
| | | MAE | 0.619○ | 0.653● | 0.620○ | 0.651● | 0.660● | 0.698● | 0.633● | 0.625 |
| Statistic | | loss/tie/win | 3/0/27 | 0/0/30 | 7/1/24 | 1/0/29 | 8/0/22 | 2/0/28 | 2/1/27 | **23/2/185*** |
| | | *p*-value | 0.001 | 0.001 | 0.005 | 0.002 | 0.005 | 0.002 | 0.003 | - |
| | | F-rank | 5.85 | 4.45 | 6.6 | 3.75 | 5.75 | 5.3 | 2.6 | **1.7** |

* The total loss/tie/win cases of MMA. ● The cases that MMA wins the other models in comparison. ○ The cases that MMA loses the comparison.

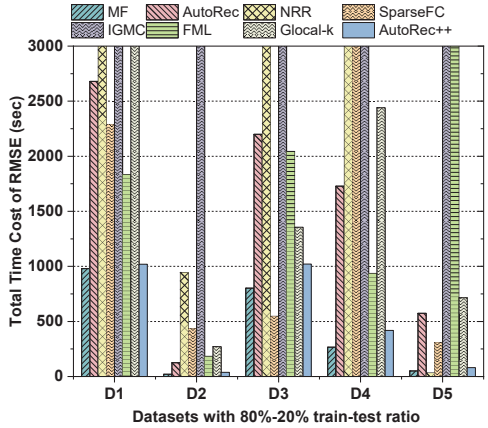
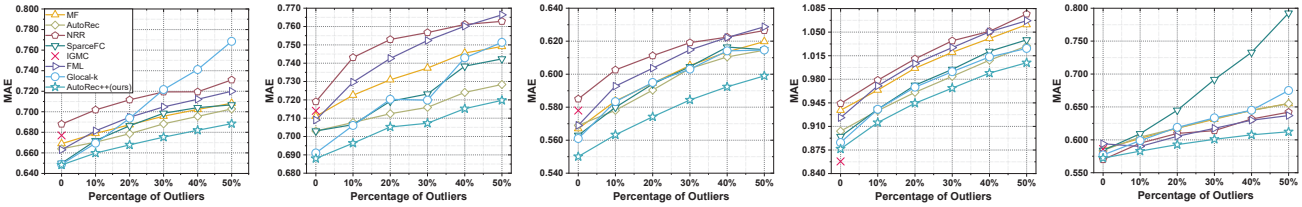



Fig. 5. The total time cost of RMSE on D1 to D5.

*3) Comparison of robustness to outliers:* To verify whether AutoRec++ can be robust to outliers by forming an $L_1$-and-$L_2$-norm-oriented loss function, we compare its performance with the competitors on the 80%-20% train-test ratio dataset with 10% to 50% outliers added. The method of adding outliers is demonstrated in Fig. S7 in Section IV of the Supplementary File to simulate the extreme cases caused by strict or malicious users. Fig. 6 and Fig. S8 in Section V of the Supplementary File record the results. From Fig. 6 and Fig. S8, we have the following observations.

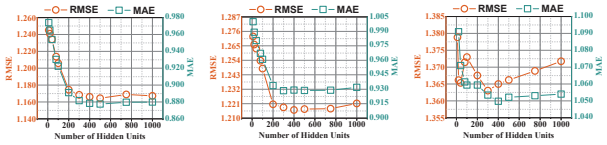
- AutoRec++ is significantly more robust than other methods. The prediction accuracy of AutoRec++ outperforms other methods almost in all datasets and under every kind of outlier percentage. But, in D4, the RMSE of AutoRec++ is slightly higher than the original model in some cases.
- Other models that do not consider biases meet severe performance degradation when facing outliers.

**Summary.** From the above observations, we conclude that AutoRec++ is robust to outliers by forming an $L_1$-and-$L_2$-norm-oriented loss function. Its robustness outperforms other both DNN-based and non-DNN-based models.




(a) D1 (b) D2 (c) D3 (d) D4 (e) D5

Fig. 6. The robustness testing results on D1 to D5 with 80%-20% train-test ratio under the MAE metric.




(a) D4 with 80%-20% train-test ratio (b) D4 with 50%-50% train-test ratio (c) D4 with 20%-80% train-test ratio

Fig. 7. The accuracy of AutoRec++ on D4 with 3 different train-test ratios as the number of Hidden Units varies.

### *D. Influence of the number of Hidden Units on AutoRec++ (RQ.3)*

Partial experimental results are shown in Fig. 7 and the remaining results are recorded in Figs. S9-S11 in SectionV of the Supplementary File demonstrate the accuracy of AutoRec++ on D1 to D5 as the number of Hidden Units varies under three kinds of train-test settings, where we have the following observations:

- More hidden units are needed with sufficient training data. The optimal number of hidden units in Fig. S9 with 80% training data is generally 700, and that of Fig. S11 is commonly 300 with 20% training data.
- Noted that an extreme case in Fig. 7(c) the resulting curve flutters up and down. However, in most other cases, setting 500 as the number of hidden units can achieve relatively optimal accuracy.

**Summary.** These observations reveal that setting the number of hidden units to 500 can obtain a globally optimal model which balances the computational cost and accuracy.

## VI. Conclusion

This paper takes the comprehensive theoretical and empirical studies in terms of the effects of different combinations of preprocessing bias (PB) and training bias (TB) as well as $L_1$-norm and $L_2$-norm on the Deep neural network (DNN)-based recommender system (RS). It proves that the DNN-based RSs' prediction accuracy and computation efficiency can be improved by incorporating PB and TB combinations. Moreover, it proposes that the self-adaptively weighted $L_1 - L_2$-norm-oriented loss function can improve Autoencoder's robustness to outliers. Empirical studies conducted on five real datasets demonstrate that: 1) the incorporation of PB and TB can significantly boost Autoencoder's prediction accuracy and computational efficiency without structural change, 2) our AutoRec++ achieves better prediction accuracy and robustness than both DNN-based and non-DNN-based state-of-the-art models, and 3) our AutoRec++ is more effective in processing sparser user behavior data. In the future, we plan to redesign the encoder layers of our AutoRec++ to enhance its debiasing effects.

## References


[1] H. Zhang, Y. Sun, M. Zhao, T. W. Chow, and Q. J. Wu, "Bridging user interest to item content for recommender systems: an optimization model," *IEEE transactions on cybernetics*, vol. 50, no. 10, pp. 4268–4280, 2019.

[2] L. Wu, X. He, X. Wang, K. Zhang, and M. Wang, "A survey on accuracy-oriented neural recommendation: From collaborative filtering to information-rich recommendation," *IEEE Transactions on Knowledge and Data Engineering*, 2022, doi: 10.1109/TKDE.2022.3145690.

[3] R. Chatterjee, S. Mazumdar, R. S. Sherratt, R. Halder, T. Maitra, and D. Giri, "Real-time speech emotion analysis for smart home assistants," *IEEE Transactions on Consumer Electronics*, vol. 67, no. 1, pp. 68–76, 2021.

[4] H. Zhang, W. Huang, L. Liu, and T. W. Chow, "Learning to match clothing from textual feature-based compatible relationships," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 11, pp. 6750–6759, 2019.

[5] M. Shang, Y. Yuan, X. Luo, and M. Zhou, "An $\alpha$–$\beta$-divergence-generalized recommender for highly accurate predictions of missing user preferences," *IEEE Transactions on Cybernetics*, vol. 52, no. 8, pp. 8006–8018, 2022.

[6] D. Wu, X. Luo, Y. He, and M. Zhou, "A prediction-sampling-based multilayer-structured latent factor model for accurate representation to high-dimensional and sparse data," *IEEE Transactions on Neural Networks and Learning Systems*, 2022, doi: 10.1109/TNNLS.2022.3200009.

[7] X. Luo, W. Qin, A. Dong, K. Sedraoui, and M. Zhou, "Efficient and high-quality recommendations via momentum-incorporated parallel stochastic gradient descent-based learning," *IEEE CAA J. Autom. Sinica*, vol. 8, no. 2, pp. 402–411, 2021.

[8] S. Zhang, L. Yao, B. Wu, X. Xu, X. Zhang, and L. Zhu, "Unraveling metric vector spaces with factorization for recommendation," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 2, pp. 732–742, 2020.

[9] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *Computer*, vol. 42, no. 8, pp. 30–37, 2009.

[10] H.-G. Kim, G. Y. Kim, and J. Y. Kim, "Music recommendation system using human activity recognition from accelerometer data," *IEEE Transactions on Consumer Electronics*, vol. 65, no. 3, pp. 349–358, 2019.

[11] P. Li, Z. Wang, Z. Ren, L. Bing, and W. Lam, "Neural rating regression with abstractive tips generation for recommendation," in *Proceedings of the 40th International ACM SIGIR conference on Research and Development in Information Retrieval*, 2017, pp. 345–354.

[12] X. Luo, H. Wu, Z. Wang, J. Wang, and D. Meng, "A novel approach to large-scale dynamically weighted directed network representation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 44, no. 12, pp. 9756–9773, 2022.

[13] X. Luo, M. Chen, H. Wu, Z. Liu, H. Yuan, and M. Zhou, "Adjusting learning depth in nonnegative latent factorization of tensors for accurately modeling temporal patterns in dynamic qos data," *IEEE Trans Autom. Sci. Eng.*, vol. 18, no. 4, pp. 2142–2155, 2021.

[14] M. Kumar, S. Gupta, X.-Z. Gao, and A. Singh, "Plant species recognition using morphological features and adaptive boosting methodology," *IEEE Access*, vol. 7, pp. 163 912–163 918, 2019.

[15] T. B. Nkwanyana and Z. Wang, "Improved particle swarm optimization base on the combination of linear decreasing and chaotic inertia weights," in *2020 12th International Conference on Computational Intelligence and Communication Networks (CICN)*. IEEE, 2020, pp. 460–465.

[16] W. Ma, M. Yu, K. Li, and G. Wang, "Why layer-wise learning is hard to scale-up and a possible solution via accelerated downsampling," in *2020 IEEE 32nd International Conference on Tools with Artificial Intelligence (ICTAI)*. IEEE, 2020, pp. 238–243.

[17] X. Luo, Z. Wang, and M. Shang, "An instance-frequency-weighted regularization scheme for non-negative latent factor analysis on high-dimensional and sparse data," *IEEE Trans. Syst. Man Cybern. Syst.*, vol. 51, no. 6, pp. 3522–3532, 2021.

[18] L. Yang, Y. Cui, Y. Xuan, C. Wang, S. Belongie, and D. Estrin, "Unbiased offline recommender evaluation for missing-not-at-random implicit feedback," in *Proceedings of the 12th ACM conference on recommender systems*, 2018, pp. 279–287.

[19] X. Luo, H. Wu, H. Yuan, and M. Zhou, "Temporal pattern-aware qos prediction via biased non-negative latent factorization of tensors," *IEEE transactions on cybernetics*, vol. 50, no. 5, pp. 1798–1809, 2019.

[20] H.-F. Yu, M. Bilenko, and C.-J. Lin, "Selection of negative samples for one-class matrix factorization," in *Proceedings of the 2017 SIAM International Conference on Data Mining*. SIAM, 2017, pp. 363–371.

[21] T. Schnabel, A. Swaminathan, A. Singh, N. Chandak, and T. Joachims, "Recommendations as treatments: Debiasing learning and evaluation," in *international conference on machine learning*. PMLR, 2016, pp. 1670–1679.

[22] D. Liang, L. Charlin, J. McInerney, and D. M. Blei, "Modeling user exposure in recommendation," in *Proceedings of the 25th international conference on World Wide Web*, 2016, pp. 951–961.

[23] Y. Yuan, X. Luo, and M.-S. Shang, "Effects of preprocessing and training biases in latent factor models for recommender systems," *Neurocomputing*, vol. 275, pp. 2019–2030, 2018.

[24] Y. Saito, "Asymmetric tri-training for debiasing missing-not-at-random explicit feedback," in *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2020, pp. 309–318.

[25] L. Muller, J. Martel, and G. Indiveri, "Kernelized synaptic weight matrices," in *International Conference on Machine Learning*. PMLR, 2018, pp. 3654–3663.

[26] S. C. Han, T. Lim, S. Long, B. Burgstaller, and J. Poon, "Glocal-k: Global and local kernels for recommender systems," in *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, 2021, pp. 3063–3067.

[27] D. Ayata, Y. Yaslan, and M. E. Kamasak, "Emotion based music recommendation system using wearable physiological sensors," *IEEE transactions on consumer electronics*, vol. 64, no. 2, pp. 196–203, 2018.

[28] S. Sedhain, A. K. Menon, S. Sanner, and L. Xie, "Autorec: Autoencoders meet collaborative filtering," in *Proceedings of the 24th international conference on World Wide Web*, 2015, pp. 111–112.

[29] S.-T. Zhong, L. Huang, C.-D. Wang, J.-H. Lai, and S. Y. Philip, "An autoencoder framework with attention mechanism for cross-domain recommendation," *IEEE Transactions on Cybernetics*, vol. 52, no. 6, pp. 5229–5241, 2022.

[30] J. Chen, H. Dong, X. Wang, F. Feng, M. Wang, and X. He, "Bias and debias in recommender system: A survey and future directions," *CoRR*, 2020. [Online]. Available: https://arxiv.org/abs/2010.03240

[31] S. Raza and C. Ding, "A regularized model to trade-off between accuracy and diversity in a news recommender system," in *2020 IEEE International Conference on Big Data (IEEE BigData 2020), Atlanta, GA, USA, December 10-13, 2020*, X. Wu, C. Jermaine, L. Xiong, X. Hu, O. Kotevska, S. Lu, W. Xu, S. Aluru, C. Zhai, E. Al-Masri, Z. Chen, and J. Saltz, Eds. IEEE, 2020, pp. 551–560.

[32] X. Zhu, X.-Y. Jing, D. Wu, Z. He, J. Cao, D. Yue, and L. Wang, "Similarity-maintaining privacy preservation and location-aware low-rank matrix factorization for qos prediction based web service recommendation," *IEEE Transactions on Services Computing*, vol. 14, no. 3, pp. 889–902, 2021.

[33] D. Wu, B. Sun, and M. Shang, "Hyperparameter learning for deep learning-based recommender systems," *IEEE Transactions on Services Computing*, 2023, doi: 10.1109/TSC.2023.3234623.

[34] X. Luo, Y. Zhou, Z. Liu, and M. Zhou, "Fast and accurate non-negative latent factor analysis of high-dimensional and sparse matrices in recommender systems," *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, no. 4, pp. 3897–3911, 2023.

[35] X. Luo, Y. Yuan, S. Chen, N. Zeng, and Z. Wang, "Position-transitional particle swarm optimization-incorporated latent factor analysis," *IEEE Transactions on Knowledge and Data Engineering*, vol. 34, no. 8, pp. 3958–3970, 2022.

[36] M. Zhang and Y. Chen, "Inductive matrix completion based on graph neural networks," in *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

[37] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec, "Graph convolutional neural networks for web-scale recommender systems," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2018, London, UK, August 19-23, 2018*, Y. Guo and F. Farooq, Eds. ACM, 2018, pp. 974–983.

[38] D. Wu, P. Zhang, Y. He, and X. Luo, "A double-space and double-norm ensembled latent factor model for highly accurate web service qos prediction," *IEEE Transactions on Services Computing*, 2022, doi: 10.1109/TSC.2022.3178543.

[39] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015.

[40] F. Monti, M. M. Bronstein, and X. Bresson, "Geometric matrix completion with recurrent multi-graph neural networks," in *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, and R. Garnett, Eds., 2017, pp. 3697–3707.

[41] J. Mendes-Moreira, C. Soares, A. M. Jorge, and J. F. de Sousa, "Ensemble approaches for regression: A survey," *ACM Comput. Surv.*, vol. 45, no. 1, pp. 10:1–10:40, 2012.

[42] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *The Journal of Machine learning research*, vol. 7, pp. 1–30, 2006.


**Teng Huang** received his Ph.D. degree in computer science in 2019 from Beihang University. Currently, he is working in the institution of Artificial Intelligence and Blockchain of Guangzhou University as Associate Professor, Guangzhou, China. He has published more than 30 academic papers, including IEEE Transactions on Dependable and Secure Computing, Information Sciences, Journal of Network and Computer Applications, International Journal of Intelligent Systems, Cognitive Computation and other journals and conferences. His research interests include Blockchain, Data security and privacy.

**Cheng Liang** *(Student Member, IEEE)* received the B.S. degree in Computer Science from Zhongkai University of Agriculture and Engineering, Guangzhou, China, in 2021. He is currently pursuing the M.S. degree in Computer Technology at Guangzhou University, Guangzhou, China. His research interests include data mining and machine learning.

**Di Wu** *(Member, IEEE)* received his Ph.D. degree from the Chongqing Institute of Green and Intelligent Technology (CIGIT), Chinese Academy of Sciences (CAS), China in 2019 and then joined CIGIT, CAS, China. He is currently a Professor of the College of Computer and Information Science, Southwest University, Chongqing, China. He has more than 60 publications, including 13 IEEE Transactions papers on T-KDE, T-NNLS, T-SC, T-SMC, and T-II, and several conferences papers on ICDM, AAAI, WWW, IJCAI, etc. His research interests include machine learning and data mining. He is serving as an Associate Editor for NEUROCOMPUTING and FRONTIERS IN NEUROROBOTICS. His homepage: https://wudi1989.github.io/Homepage/

**Yi He** *(Member, IEEE)* received his Ph.D. degree in computer science from the University of Louisiana at Lafayette and a B.E. from the Harbin Institute of Technology (China) in 2020 and 2013, respectively. Dr. Yi He is an assistant professor of Computer Science at Old Dominion University, USA. His research focus lies broadly in data mining and machine learning. His research outcomes have appeared in top venues, e.g., AAAI, IJCAI, WWW, ICDM, SDM, TKDE, TNNLS, to name a few. He has served as multiple roles in the data mining and machine learning community, including the registration chair of ICDM 2022, session chair of ICMR 2022, and (T)PC member of AAAI'20,'21,'22, IJCAI'22, ICDM'22, CIKM'22, ECML-PKDD'22, and referees for prestigious journal such as TNNLS, TMC, and TITS. His homepage: https://www.lions.odu.edu/~y1he/index.html